

School Management System Project Report

Object Oriented Programming



December 22, 2024

majid farooq 045

HAbib ur rehman 116

**School Management System Project Report**

**1. Introduction**

The **School Management System** (SMS) is designed to automate and simplify the management of students, teachers, administrative staff, and courses in an educational institution. The project has Java's Object-Oriented Programming (OOP) concepts such as inheritance, polymorphism, interfaces, and file handling to provide a seamless experience for managing various operations like course enrollment, report generation, and data persistence.

This report outlines the features, design, and implementation of the School Management System, developed by a team for the Semester Project.

**2. Project Overview**

The **School Management System** will manage the following components:

* **Students**: Students can enroll in courses, view their enrolled courses, and maintain their personal details.
* **Teachers**: Teachers are assigned courses to teach, and they can view the courses they teach.
* **Administrative Staff**: Staff can generate reports, add or remove students, teachers, and courses, and manage the overall operations of the system.
* **Courses**: Courses can have students enrolled, teachers assigned, and grades tracked.

The system integrates a **Graphical User Interface (GUI)** for managing students, teachers, and courses, allowing users to perform CRUD (Create, Read, Update, Delete) operations.

**3. Functions**

**Core Functionalities**

1. **User Roles**
   * **Student**:
     + Attributes: studentID, name, address, dateOfBirth, list of enrolled courses.
     + Methods:
       - enrollInCourse(Course course): Adds a course to the student's list of enrolled courses.
       - displayCourses(): Lists all the courses the student is enrolled in.
   * **Teacher**:
     + Attributes: teacherID, name, specialization, list of courses taught.
     + Methods:
       - assignCourse(Course course): Adds a course to the teacher's list of courses.
       - displayCourses(): Lists all the courses taught by the teacher.
   * **Administrative Staff**:
     + Attributes: staffID, name, role, and department.
     + Methods:
       - generateReport(List<Person> people): Generates reports for students, teachers, or courses.
2. **Course Management**:
   * Attributes: courseID, title, credits, assignedTeacher, list of enrolled students.
   * Methods:
     + addStudent(Student student): Adds a student to the course.
     + removeStudent(Student student): Removes a student from the course.
     + calculateAverageGrade(): Calculates the average grade of all students in the course.
3. **Inheritance**:
   * Base class Person containing common attributes (name, email, dateOfBirth).
   * Student, Teacher, and AdministrativeStaff classes inherit from Person.
4. **Polymorphism and Dynamic Binding**:
   * Overridden methods such as generateReport() in AdministrativeStaff and displayDetails() in Student and Teacher.
5. **Static Data Members and Methods**:
   * Static counters to track the total number of students, teachers, and courses In University class.
   * Static method displaySystemStats() to show system statistics.
6. **Interfaces**:
   * Interface Reportable with methods generateReport() and exportToFile(), implemented by AdministrativeStaff and Teacher.
7. **Composition**:
   * A Course object references a Teacher object and an array of Student objects.
8. **Object Arrays and ArrayList**:
   * Use of object arrays to store Student and Course objects.
   * Use of ArrayList for dynamic management of students enrolled in courses.
9. **File Handling**:
   * Methods to load and save data to files: loadData(String filename) and saveData(String filename) using buffer and input output stream
10. **Generics**:
    * Generic class Repository<T> for managing lists of Student, Teacher, and Course.
11. **Wrapper Classes**:
    * Use of wrapper classes for numeric data processing, such as calculating the average and median grades in courses.
12. **Static & Dynamic Typing**:
    * Demonstrating static typing with explicit type declarations.
    * Dynamic typing with object references and downcasting.

**4. Design**

**Class Diagram**

The system consists of the following main classes:

* **Person (Base Class)**: Attributes: name, email, dateOfBirth.
  + **Student**: Inherits from Person, adds studentID, enrolledCourses, methods like enrollInCourse() and displayCourses().
  + **Teacher**: Inherits from Person, adds teacherID, specialization, assignedCourses, methods like assignCourse() and displayCourses().
  + **AdministrativeStaff**: Inherits from Person, adds staffID, role, department, methods like generateReport().
* **Course**: Attributes: courseID, title, credits, assignedTeacher, students, methods like addStudent(), removeStudent(), calculateAverageGrade().
* **University**: Manages collections of students, teachers, and courses having static listss, handles data loading and saving.
* **Repository**: A generic class to manage lists of Student, Teacher, and Course objects.
* **Reportable**: Interface defining methods generateReport() and exportToFile().

**GUI Design**

The system includes the following GUI components:

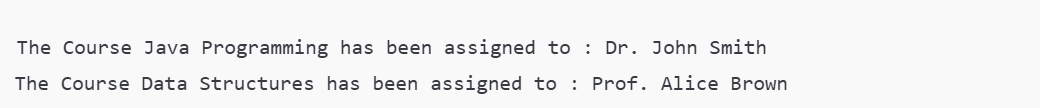
* **Main Menu**: Buttons for navigating between the student, teacher, and administrative interfaces.
* **Student Panel**: Allows students to enroll in courses and view their enrolled courses.
* **Teacher Panel**: Allows teachers to assign courses and view the courses they are teaching.
* **Administrative Panel**: Allows staff to generate reports and manage the overall system.
* **Report Generation**: Option to generate reports for students, teachers, or courses.

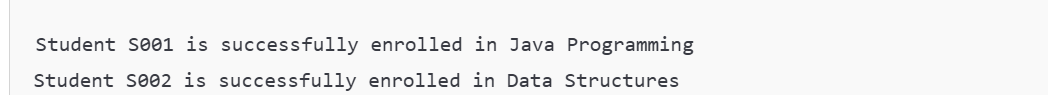
**5. Implementation**

**Core Concepts Used**

1. **Inheritance**: The base class Person allows the classes Student, Teacher, and AdministrativeStaff to inherit common properties and methods, reducing redundancy.
2. **Polymorphism**: The generateReport() and displayDetails() methods demonstrate polymorphism by providing class-specific implementations for generating reports and displaying information.
3. **Composition**: The Course class is composed of a Teacher object and an array of Student objects, emphasizing real-world relationships.
4. **File Handling**: The system uses file handling to save and retrieve data, ensuring persistence of student, teacher, and course records.

Sample Output without gui





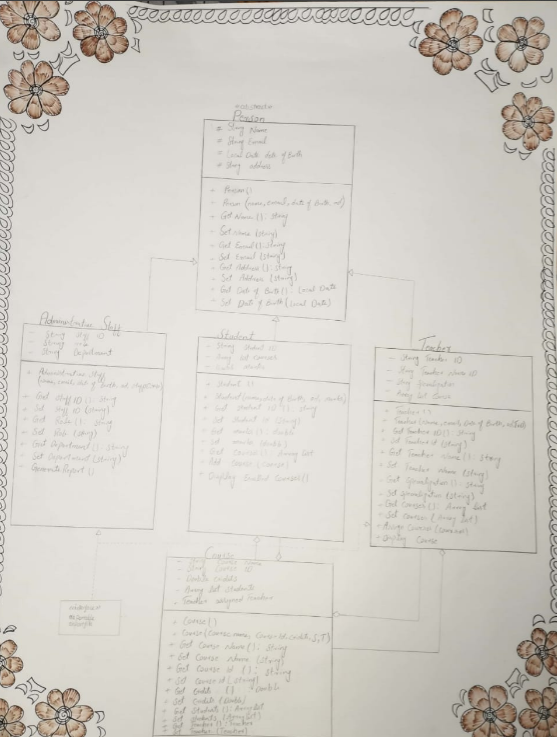
A screen shot of a computer

Description automatically generated

A close-up of a website

Description automatically generated

**Class Diagram**



**Implementation Code**

Crud GUI

import javax.swing.\*;

import java.awt.\*;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.time.LocalDate;

import java.util.ArrayList;

public class crud {

    private JPanel dynamicPanel;

    public crud() {

        JFrame frame = new JFrame("CRUD Application");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.setSize(600, 400);

        frame.setLayout(new BorderLayout());

        // Menu bar setup

        JMenuBar menuBar = new JMenuBar();

        JMenu studentMenu = new JMenu("Student");

        JMenuItem addStudentMenuItem = new JMenuItem("Add Student");

        JMenuItem viewStudentMenuItem = new JMenuItem("View Student by ID");

        JMenuItem updateStudentMenuItem = new JMenuItem("Update Student by ID");

        JMenuItem deleteStudentMenuItem = new JMenuItem("Delete Student by ID");

        studentMenu.add(addStudentMenuItem);

        studentMenu.add(viewStudentMenuItem);

        studentMenu.add(updateStudentMenuItem);

        studentMenu.add(deleteStudentMenuItem);

        JMenu teacherMenu = new JMenu("Teacher");

        JMenuItem addTeacherMenuItem = new JMenuItem("Add Teacher");

        JMenuItem viewTeacherMenuItem = new JMenuItem("View Teacher by ID");

        JMenuItem updateTeacherMenuItem = new JMenuItem("Update Teacher by ID");

        JMenuItem deleteTeacherMenuItem = new JMenuItem("Delete Teacher by ID");

        teacherMenu.add(addTeacherMenuItem);

        teacherMenu.add(viewTeacherMenuItem);

        teacherMenu.add(updateTeacherMenuItem);

        teacherMenu.add(deleteTeacherMenuItem);

        JMenu courseMenu = new JMenu("Course");

        JMenuItem addCourseMenuItem = new JMenuItem("Add Course");

        JMenuItem viewCourseMenuItem = new JMenuItem("View Course by ID");

        JMenuItem updateCourseMenuItem = new JMenuItem("Update Course by ID");

        JMenuItem deleteCourseMenuItem = new JMenuItem("Delete Course by ID");

        courseMenu.add(addCourseMenuItem);

        courseMenu.add(viewCourseMenuItem);

        courseMenu.add(updateCourseMenuItem);

        courseMenu.add(deleteCourseMenuItem);

        menuBar.add(studentMenu);

        menuBar.add(teacherMenu);

        menuBar.add(courseMenu);

        frame.setJMenuBar(menuBar);

        dynamicPanel = new JPanel();

        dynamicPanel.setLayout(new BoxLayout(dynamicPanel, BoxLayout.Y\_AXIS));

        frame.add(dynamicPanel, BorderLayout.CENTER);

        // Student menu actions

        addStudentMenuItem.addActionListener(e -> showAddStudentForm());

        viewStudentMenuItem.addActionListener(e -> handleViewStudent());

        updateStudentMenuItem.addActionListener(e -> handleUpdateStudent());

        deleteStudentMenuItem.addActionListener(e -> handleDeleteStudent());

        // Teacher menu actions

        addTeacherMenuItem.addActionListener(e -> showAddTeacherForm());

        viewTeacherMenuItem.addActionListener(e -> handleViewTeacher());

        updateTeacherMenuItem.addActionListener(e -> handleUpdateTeacher());

        deleteTeacherMenuItem.addActionListener(e -> handleDeleteTeacher());

        // Course menu actions

        addCourseMenuItem.addActionListener(e -> handleAddCourse());

        viewCourseMenuItem.addActionListener(e -> handleViewCourse());

        updateCourseMenuItem.addActionListener(e -> handleUpdateCourse());

        deleteCourseMenuItem.addActionListener(e -> handleDeleteCourse());

        frame.setVisible(true);

    }

    // Course methods

    private void handleAddCourse() {

        dynamicPanel.removeAll();

        JLabel courseNameLabel = new JLabel("Course Name:");

        JTextField courseNameInput = new JTextField(20);

        JLabel courseIdLabel = new JLabel("Course ID:");

        JTextField courseIdInput = new JTextField(20);

        JLabel courseCreditsLabel = new JLabel("Course Credits:");

        JTextField courseCreditsInput = new JTextField(20);

        JButton saveButton = new JButton("Save");

        dynamicPanel.add(courseNameLabel);

        dynamicPanel.add(courseNameInput);

        dynamicPanel.add(courseIdLabel);

        dynamicPanel.add(courseIdInput);

        dynamicPanel.add(courseCreditsLabel);

        dynamicPanel.add(courseCreditsInput);

        dynamicPanel.add(saveButton);

        dynamicPanel.revalidate();

        dynamicPanel.repaint();

        saveButton.addActionListener(e -> {

            try {

                String courseName = courseNameInput.getText();

                String courseId = courseIdInput.getText();

                int courseCredits = Integer.parseInt(courseCreditsInput.getText());

                if (courseName.isEmpty() || courseId.isEmpty()) {

                    JOptionPane.showMessageDialog(null, "All fields are required!", "Error", JOptionPane.ERROR\_MESSAGE);

                    return;

                }

                Course course = new Course(courseName, courseId, courseCredits, null, null);

                University.saveData("university\_data.txt");

                JOptionPane.showMessageDialog(null, "Course added successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

                courseNameInput.setText("");

                courseIdInput.setText("");

                courseCreditsInput.setText("");

            } catch (Exception ex) {

                JOptionPane.showMessageDialog(null, "Invalid input: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

            }

        });

    }

    private void handleViewCourse() {

        String courseId = JOptionPane.showInputDialog("Enter Course ID to view:");

        if (courseId != null && !courseId.trim().isEmpty()) {

            ArrayList<Course> courseList = University.getCourseList();

            for (Course course : courseList) {

                if (course.getCourseId().equals(courseId)) {

                    JOptionPane.showMessageDialog(null, course.toString(), "Course Details", JOptionPane.INFORMATION\_MESSAGE);

                    return;

                }

            }

            JOptionPane.showMessageDialog(null, "Course not found!", "Error", JOptionPane.ERROR\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid Course ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void handleUpdateCourse() {

        String courseId = JOptionPane.showInputDialog("Enter Course ID to update:");

        if (courseId != null && !courseId.trim().isEmpty()) {

            ArrayList<Course> courseList = University.getCourseList();

            for (Course course : courseList) {

                if (course.getCourseId().equals(courseId)) {

                    // Show the course update form

                    dynamicPanel.removeAll();

                    JLabel courseNameLabel = new JLabel("Course Name:");

                    JTextField courseNameInput = new JTextField(course.getCourseName(), 20);

                    JLabel courseIdLabel = new JLabel("Course ID:");

                    JTextField courseIdInput = new JTextField(course.getCourseId(), 20);

                    JLabel courseCreditsLabel = new JLabel("Course Credits:");

                    JTextField courseCreditsInput = new JTextField(String.valueOf(course.getCredits()), 20);

                    JButton updateButton = new JButton("Update");

                    dynamicPanel.add(courseNameLabel);

                    dynamicPanel.add(courseNameInput);

                    dynamicPanel.add(courseIdLabel);

                    dynamicPanel.add(courseIdInput);

                    dynamicPanel.add(courseCreditsLabel);

                    dynamicPanel.add(courseCreditsInput);

                    dynamicPanel.add(updateButton);

                    dynamicPanel.revalidate();

                    dynamicPanel.repaint();

                    updateButton.addActionListener(updateEvent -> {

                        try {

                            course.setCourseName(courseNameInput.getText());

                            course.setCourseId(courseIdInput.getText());

                            course.setCredits(Integer.parseInt(courseCreditsInput.getText()));

                            JOptionPane.showMessageDialog(null, "Course updated successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

                        } catch (Exception ex) {

                            JOptionPane.showMessageDialog(null, "Invalid input: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

                        }

                    });

                    return;

                }

            }

            JOptionPane.showMessageDialog(null, "Course not found!", "Error", JOptionPane.ERROR\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid Course ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void handleDeleteCourse() {

    String courseId = JOptionPane.showInputDialog("Enter Course ID to delete:");

    if (courseId != null && !courseId.trim().isEmpty()) {

        ArrayList<Course> courseList = University.getCourseList();

        boolean found = false;

        // Loop through the course list to find and delete the course

        for (int i = 0; i < courseList.size(); i++) {

            if (courseList.get(i).getCourseId().equals(courseId)) {

                courseList.remove(i);

                found = true;

                break;

            }

        }

        // Show appropriate message based on whether the course was found and deleted

        if (found) {

            // Update the course list in University

            University.setCourseList(courseList);

            // Write the updated list back to the file

            writeCourseListToFile(courseList);

            JOptionPane.showMessageDialog(null, "Course deleted successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Course ID not found.", "Error", JOptionPane.ERROR\_MESSAGE);

        }

    } else {

        JOptionPane.showMessageDialog(null, "Please enter a valid Course ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

    }

}

private void writeCourseListToFile(ArrayList<Course> courseList) {

    try (BufferedWriter writer = new BufferedWriter(new FileWriter("university\_data.txt"))) {

        University.setCourseList(courseList);

        for (Course course : courseList) {

            // Assuming Course has a method to represent its data in a suitable format

            writer.write(course.getCourseName() + "|" + course.getCourseId() + "|" + course.getCredits());

            writer.newLine();

        }

    } catch (IOException e) {

        JOptionPane.showMessageDialog(null, "Error updating course file.", "Error", JOptionPane.ERROR\_MESSAGE);

        e.printStackTrace();

    }

}

    // Teacher methods

    private void showAddTeacherForm() {

        showFormForTeacher("Teacher", "Name", "Email", "Teacher ID",

                           "Date of Birth (YYYY-MM-DD)", "Address", "Specialization");

    }

    private void showFormForTeacher(String entity, String field1Label, String field2Label, String field3Label,

                                    String field4Label, String field5Label, String field6Label) {

        dynamicPanel.removeAll();

        // Input fields

        JLabel field1 = new JLabel(field1Label + ":");

        JTextField field1Input = new JTextField(20);

        JLabel field2 = new JLabel(field2Label + ":");

        JTextField field2Input = new JTextField(20);

        JLabel field3 = new JLabel(field3Label + ":");

        JTextField field3Input = new JTextField(20);

        JLabel field4 = new JLabel(field4Label + ":");

        JTextField field4Input = new JTextField(20);

        JLabel field5 = new JLabel(field5Label + ":");

        JTextField field5Input = new JTextField(20);

        JLabel field6 = new JLabel(field6Label + ":");

        JTextField field6Input = new JTextField(20);

        JButton saveButton = new JButton("Save");

        // Add inputs to the panel

        dynamicPanel.add(field1);

        dynamicPanel.add(field1Input);

        dynamicPanel.add(field2);

        dynamicPanel.add(field2Input);

        dynamicPanel.add(field3);

        dynamicPanel.add(field3Input);

        dynamicPanel.add(field4);

        dynamicPanel.add(field4Input);

        dynamicPanel.add(field5);

        dynamicPanel.add(field5Input);

        dynamicPanel.add(field6);

        dynamicPanel.add(field6Input);

        dynamicPanel.add(saveButton);

        dynamicPanel.revalidate();

        dynamicPanel.repaint();

        saveButton.addActionListener(e -> {

            try {

                String name = field1Input.getText();

                String email = field2Input.getText();

                String teacherId = field3Input.getText();

                LocalDate dateOfBirth = LocalDate.parse(field4Input.getText());

                String address = field5Input.getText();

                String specialization = field6Input.getText();

                if (name.isEmpty() || email.isEmpty() || teacherId.isEmpty() || address.isEmpty() || specialization.isEmpty()) {

                    JOptionPane.showMessageDialog(null, "All fields are required!", "Error", JOptionPane.ERROR\_MESSAGE);

                    return;

                }

                ArrayList<Course> courses = new ArrayList<>(); // Default courses

                Teacher teacher = new Teacher(name, email, dateOfBirth, address, teacherId, name, specialization, courses);

                University.saveData("university\_data.txt");

                ArrayList<Teacher> teachers = University.getTeacherList();

                teachers.add(teacher);

                University.setTeacherList(teachers);

                JOptionPane.showMessageDialog(null, entity + " added successfully!", "Success",

                        JOptionPane.INFORMATION\_MESSAGE);

                // Clear input fields

                field1Input.setText("");

                field2Input.setText("");

                field3Input.setText("");

                field4Input.setText("");

                field5Input.setText("");

                field6Input.setText("");

            } catch (Exception ex) {

                JOptionPane.showMessageDialog(null, "Invalid input: " + ex.getMessage(), "Error",

                        JOptionPane.ERROR\_MESSAGE);

            }

        });

    }

    private void handleViewTeacher() {

        String id = JOptionPane.showInputDialog("Enter Teacher ID to view:");

        if (id != null && !id.trim().isEmpty()) {

            ArrayList<Teacher> teachers = University.getTeacherList();

            for (Teacher teacher : teachers) {

                if (teacher.getTeacherId().equals(id)) {

                    JOptionPane.showMessageDialog(null, teacher.toString(), "Teacher Details",

                            JOptionPane.INFORMATION\_MESSAGE);

                    return;

                }

            }

            JOptionPane.showMessageDialog(null, "Teacher not found!", "Error", JOptionPane.ERROR\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void handleUpdateTeacher() {

        String id = JOptionPane.showInputDialog("Enter Teacher ID to update:");

        if (id != null && !id.trim().isEmpty()) {

            ArrayList<Teacher> teachers = University.getTeacherList();

            for (Teacher teacher : teachers) {

                if (teacher.getTeacherId().equals(id)) {

                    // Show the teacher update form

                    dynamicPanel.removeAll();

                    JLabel nameLabel = new JLabel("Name:");

                    JTextField nameInput = new JTextField(teacher.getName(), 20);

                    JLabel emailLabel = new JLabel("Email:");

                    JTextField emailInput = new JTextField(teacher.getEmail(), 20);

                    JLabel teacherIdLabel = new JLabel("Teacher ID:");

                    JTextField teacherIdInput = new JTextField(teacher.getTeacherId(), 20);

                    JLabel addressLabel = new JLabel("Address:");

                    JTextField addressInput = new JTextField(teacher.getAddress(), 20);

                    JLabel specializationLabel = new JLabel("Specialization:");

                    JTextField specializationInput = new JTextField(teacher.getSpecialization(), 20);

                    JButton updateButton = new JButton("Update");

                    dynamicPanel.add(nameLabel);

                    dynamicPanel.add(nameInput);

                    dynamicPanel.add(emailLabel);

                    dynamicPanel.add(emailInput);

                    dynamicPanel.add(teacherIdLabel);

                    dynamicPanel.add(teacherIdInput);

                    dynamicPanel.add(addressLabel);

                    dynamicPanel.add(addressInput);

                    dynamicPanel.add(specializationLabel);

                    dynamicPanel.add(specializationInput);

                    dynamicPanel.add(updateButton);

                    dynamicPanel.revalidate();

                    dynamicPanel.repaint();

                    // Add action listener for the update button

                    updateButton.addActionListener(updateEvent -> {

                        try {

                            teacher.setName(nameInput.getText());

                            teacher.setEmail(emailInput.getText());

                            teacher.setTeacherId(teacherIdInput.getText());

                            teacher.setAddress(addressInput.getText());

                            teacher.setSpecialization(specializationInput.getText());

                            JOptionPane.showMessageDialog(null, "Teacher updated successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

                            // After update, make sure to save the updated list (if necessary)

                            University.setTeacherList(teachers); // If needed

                        } catch (Exception ex) {

                            JOptionPane.showMessageDialog(null, "Invalid input: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

                        }

                    });

                    return;

                }

            }

            JOptionPane.showMessageDialog(null, "Teacher not found!", "Error", JOptionPane.ERROR\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid Teacher ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void handleDeleteTeacher() {

        String id = JOptionPane.showInputDialog("Enter Teacher ID to delete:");

        if (id != null && !id.trim().isEmpty()) {

            ArrayList<Teacher> teachers = University.getTeacherList();

            boolean teacherFound = false;

            // Remove the teacher from the list

            for (int i = 0; i < teachers.size(); i++) {

                if (teachers.get(i).getTeacherId().equals(id)) {

                    teachers.remove(i);

                    teacherFound = true;

                    break;

                }

            }

            if (teacherFound) {

                // Update the list in University class

                // You might not need to set it again; just modify the existing list

                University.setTeacherList(teachers);  // You can directly modify the static list

                // Rewrite the updated list to the file

                updateTeacherFile(teachers);

                JOptionPane.showMessageDialog(null, "Teacher deleted successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

            } else {

                JOptionPane.showMessageDialog(null, "Teacher ID not found!", "Error", JOptionPane.ERROR\_MESSAGE);

            }

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void updateTeacherFile(ArrayList<Teacher> teachers) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter("university\_data.txt"))) {

            for (Teacher teacher : teachers) {

                // Assuming the file format for each teacher is: teacherId, name, specialization

                writer.write(teacher.getTeacherId() + "," + teacher.getName() + "," + teacher.getSpecialization());

                writer.newLine();

            }

        } catch (IOException e) {

            JOptionPane.showMessageDialog(null, "Error updating teacher file.", "Error", JOptionPane.ERROR\_MESSAGE);

            e.printStackTrace();

        }

    }

    private void showAddStudentForm() {

        showForm("Student", "Name", "Email", "Student ID", "Date of Birth (YYYY-MM-DD)", "Address", "Marks");

    }

    private void handleViewStudent() {

        String id = JOptionPane.showInputDialog("Enter Student ID to view:");

        if (id != null && !id.trim().isEmpty()) {

            ArrayList<Student> students = University.getStudentList();

            for (Student student : students) {

                if (student.getStudentId().equals(id)) {

                    JOptionPane.showMessageDialog(null, student.toString(), "Student Details",

                            JOptionPane.INFORMATION\_MESSAGE);

                    return;

                }

            }

            JOptionPane.showMessageDialog(null, "Student not found!", "Error", JOptionPane.ERROR\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void handleUpdateStudent() {

        String id = JOptionPane.showInputDialog("Enter Student ID to update:");

        if (id != null && !id.trim().isEmpty()) {

            ArrayList<Student> students = University.getStudentList();

            for (Student student : students) {

                if (student.getStudentId().equals(id)) {

                    // Show the student update form

                    dynamicPanel.removeAll();

                    JLabel nameLabel = new JLabel("Name:");

                    JTextField nameInput = new JTextField(student.getName(), 20);

                    JLabel emailLabel = new JLabel("Email:");

                    JTextField emailInput = new JTextField(student.getEmail(), 20);

                    JLabel studentIdLabel = new JLabel("Student ID:");

                    JTextField studentIdInput = new JTextField(student.getStudentId(), 20);

                    JLabel addressLabel = new JLabel("Address:");

                    JTextField addressInput = new JTextField(student.getAddress(), 20);

                    JLabel marksLabel = new JLabel("Marks:");

                    JTextField marksInput = new JTextField(String.valueOf(student.getMarks()), 20);

                    JButton updateButton = new JButton("Update");

                    dynamicPanel.add(nameLabel);

                    dynamicPanel.add(nameInput);

                    dynamicPanel.add(emailLabel);

                    dynamicPanel.add(emailInput);

                    dynamicPanel.add(studentIdLabel);

                    dynamicPanel.add(studentIdInput);

                    dynamicPanel.add(addressLabel);

                    dynamicPanel.add(addressInput);

                    dynamicPanel.add(marksLabel);

                    dynamicPanel.add(marksInput);

                    dynamicPanel.add(updateButton);

                    dynamicPanel.revalidate();

                    dynamicPanel.repaint();

                    // Add action listener for the update button

                    updateButton.addActionListener(updateEvent -> {

                        try {

                            student.setName(nameInput.getText());

                            student.setEmail(emailInput.getText());

                            student.setStudentId(studentIdInput.getText());

                            student.setAddress(addressInput.getText());

                            student.setMarks(Double.parseDouble(marksInput.getText()));

                            JOptionPane.showMessageDialog(null, "Student updated successfully!", "Success", JOptionPane.INFORMATION\_MESSAGE);

                            // After update, save the updated student list (if needed)

                            University.setStudentList(students); // If needed

                        } catch (Exception ex) {

                            JOptionPane.showMessageDialog(null, "Invalid input: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

                        }

                    });

                    return;

                }

            }

            JOptionPane.showMessageDialog(null, "Student not found!", "Error", JOptionPane.ERROR\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid Student ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void handleDeleteStudent() {

        String id = JOptionPane.showInputDialog("Enter Student ID to delete:");

        if (id != null && !id.trim().isEmpty()) {

            ArrayList<Student> students = University.getStudentList();

            students.removeIf(student -> student.getStudentId().equals(id));

            University.setStudentList(students);

            JOptionPane.showMessageDialog(null, "Student deleted successfully!", "Success",

                    JOptionPane.INFORMATION\_MESSAGE);

        } else {

            JOptionPane.showMessageDialog(null, "Please enter a valid ID.", "Warning", JOptionPane.WARNING\_MESSAGE);

        }

    }

    private void showForm(String entity, String field1Label, String field2Label, String field3Label,

                          String field4Label, String field5Label, String field6Label) {

        dynamicPanel.removeAll();

        // Input fields

        JLabel field1 = new JLabel(field1Label + ":");

        JTextField field1Input = new JTextField(20);

        JLabel field2 = new JLabel(field2Label + ":");

        JTextField field2Input = new JTextField(20);

        JLabel field3 = new JLabel(field3Label + ":");

        JTextField field3Input = new JTextField(20);

        JLabel field4 = new JLabel(field4Label + ":");

        JTextField field4Input = new JTextField(20);

        JLabel field5 = new JLabel(field5Label + ":");

        JTextField field5Input = new JTextField(20);

        JLabel field6 = new JLabel(field6Label + ":");

        JTextField field6Input = new JTextField(20);

        JButton saveButton = new JButton("Save");

        // Add inputs to the panel

        dynamicPanel.add(field1);

        dynamicPanel.add(field1Input);

        dynamicPanel.add(field2);

        dynamicPanel.add(field2Input);

        dynamicPanel.add(field3);

        dynamicPanel.add(field3Input);

        dynamicPanel.add(field4);

        dynamicPanel.add(field4Input);

        dynamicPanel.add(field5);

        dynamicPanel.add(field5Input);

        dynamicPanel.add(field6);

        dynamicPanel.add(field6Input);

        dynamicPanel.add(saveButton);

        dynamicPanel.revalidate();

        dynamicPanel.repaint();

        saveButton.addActionListener(e -> {

            try {

                String name = field1Input.getText();

                String email = field2Input.getText();

                String studentId = field3Input.getText();

                LocalDate dateOfBirth = LocalDate.parse(field4Input.getText());

                String address = field5Input.getText();

                double marks = Double.parseDouble(field6Input.getText());

                if (name.isEmpty() || email.isEmpty() || studentId.isEmpty() || address.isEmpty()) {

                    JOptionPane.showMessageDialog(null, "All fields are required!", "Error", JOptionPane.ERROR\_MESSAGE);

                    return;

                }

                ArrayList<Course> courses = new ArrayList<>();

                Student student = new Student(name, email, dateOfBirth, address, studentId, marks, courses);

                ArrayList<Student> students = University.getStudentList();

                students.add(student);

                University.setStudentList(students);

                JOptionPane.showMessageDialog(null, entity + " added successfully!", "Success",

                        JOptionPane.INFORMATION\_MESSAGE);

                // Clear input fields

                field1Input.setText("");

                field2Input.setText("");

                field3Input.setText("");

                field4Input.setText("");

                field5Input.setText("");

                field6Input.setText("");

            } catch (Exception ex) {

                JOptionPane.showMessageDialog(null, "Invalid input: " + ex.getMessage(), "Error",

                        JOptionPane.ERROR\_MESSAGE);

            }

        });

    }

    public static void main(String[] args) {

        ArrayList<Student> students = new ArrayList<>();

            ArrayList<Teacher> teachers = new ArrayList<>();

            ArrayList<AdministrativeStaff> staff = new ArrayList<>();

            ArrayList<Course> courses = new ArrayList<>();

            University university = new University("Tech Campus", students, teachers, staff, courses);

            new crud();

        University.saveData("university\_data.txt");

        ArrayList<?> dataList = University.loadData("university\_data.txt"); // Load data as a generic list

          // Separate the data into specific lists

          ArrayList<Student> student1 = new ArrayList<>();

          ArrayList<Teacher> teacher1 = new ArrayList<>();

          ArrayList<AdministrativeStaff> staffs = new ArrayList<>();

          ArrayList<Course> course1 = new ArrayList<>();

          // Iterate through the loaded dataList and separate them by type

          for (Object item : dataList) {

              if (item instanceof Student) {

                  student1.add((Student) item);

              } else if (item instanceof Teacher) {

                  teacher1.add((Teacher) item);

              } else if (item instanceof AdministrativeStaff) {

                  staffs.add((AdministrativeStaff) item);

              } else if (item instanceof Course) {

                  course1.add((Course) item);

              }

          }

          // Now, you can use these lists to process or display the data as needed

          System.out.println("Students:");

          for (Student student : student1) {

              System.out.println(student);

          }

          System.out.println("Teachers:");

          for (Teacher teacher : teacher1) {

              System.out.println(teacher);

          }

          System.out.println("Administrative Staff:");

          for (AdministrativeStaff admin : staffs) {

              System.out.println(admin);

          }

          System.out.println("Courses:");

          for (Course course : course1) {

              System.out.println(course);

          }

    }

}

SEmproject1.java file code

// Create a base class Person with common attributes like name, email, and dateOfBirth. Use

// inheritance to create Student, Teacher, and AdministrativeStaff classes.

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.Collections;

interface Reportable {

    void generateReport();

}

class Repository<T> {

    private ArrayList<T> items;

    public Repository() {

        this.items = new ArrayList<>();

    }

    // Add item to the repository and save to file

    public void add(T item) {

        items.add(item);

        System.out.println("Added: " + item);

    }

    public ArrayList<T> getAll() {

        return new ArrayList<>(items);

    }

}

abstract class person {

    protected String name;

    protected String email;

    protected LocalDate dateOfBirth;

    protected String address;

    public person() {

    }

    public person(String name, String email, LocalDate dateOfBirth, String ad) {

        this.name = name;

        this.email = email;

        this.dateOfBirth = dateOfBirth;

        this.address = ad;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getEmail() {

        return email;

    }

    public String getAddress() {

        return address;

    }

    public void setEmail(String email) {

        this.email = email;

    }

    public LocalDate getDateOfBirth() {

        return dateOfBirth;

    }

    public void setDateOfBirth(LocalDate dateOfBirth) {

        if (dateOfBirth.isAfter(LocalDate.now())) {

            throw new IllegalArgumentException("Date of birth cannot be in the future");

        }

        this.dateOfBirth = dateOfBirth;

    }

    public void setAddress(String address) {

        this.address = address;

    }

    @Override

    public String toString() {

        return "person [name=" + name + ", email=" + email + ", dateOfBirth=" + dateOfBirth + "]";

    }

}

class Student extends person {

    private String StudentId;

    private ArrayList<Course> courses;

    private double marks;

    public Student() {

        University.StudentList.add(this);

    }

    public Student(String name, String email, LocalDate dateOfBirth, String ad, String studentId, double marks,

            ArrayList<Course> courses) {

        super(name, email, dateOfBirth, ad);

        StudentId = studentId;

        this.courses = courses;

        this.marks = marks;

        University.StudentList.add(this);

    }

    public String getStudentId() {

        return StudentId;

    }

    public double getMarks() {

        return marks;

    }

    public void setMarks(double marks) {

        this.marks = marks;

    }

    public void setStudentId(String studentId) {

        StudentId = studentId;

    }

    public ArrayList<Course> getCourses() {

        return courses;

    }

    public void setCourses(ArrayList<Course> courses) {

        this.courses = courses;

    }

    @Override

    public String toString() {

        return "Student [name=" + name + ", email=" + email + ", StudentId=" + StudentId + ", marks=" + marks + "]";

    }

    public void addCourse(Course course) {

        this.courses.add(course);

        System.out.println("Student " + this.StudentId + " is successfully enrolled in " + course.getCourseName());

    }

    // Method to display all enrolled courses

    public void displayEnrolledCourses() {

        System.out.println(

                this.getName() + " whose ID number is " + this.StudentId + " is enrolled in the following courses:");

        for (Course course : courses) {

            System.out.println(course.getCourseName() + " with Credit Hours: " + course.getCredits());

        }

    }

}

class Course {

    // Attributes: courseID, title, credits, assignedTeacher, list of enrolled

    // students.

    private String courseName;

    private String courseId;

    private double credits;

    private ArrayList<Student> students;

    private Teacher assignedteacher;

    public Course() {

    }

    public Course(String courseName, String courseId, double credits, ArrayList<Student> students, Teacher teacher) {

        this.courseName = courseName;

        this.courseId = courseId;

        this.credits = credits;

        this.students = students;

        this.assignedteacher = teacher;

        University.CourseList.add(this);

    }

    public String getCourseName() {

        return courseName;

    }

    public void setCourseName(String courseName) {

        this.courseName = courseName;

    }

    public String getCourseId() {

        return courseId;

    }

    public void setCourseId(String courseId) {

        this.courseId = courseId;

    }

    public double getCredits() {

        return credits;

    }

    public void setCredits(double credits) {

        this.credits = credits;

    }

    public ArrayList<Student> getStudents() {

        return students;

    }

    public void setStudents(ArrayList<Student> students) {

        this.students = students;

    }

    public Teacher getTeacher() {

        return assignedteacher;

    }

    public void setTeacher(Teacher teacher) {

        this.assignedteacher = teacher;

    }

    public void addStudent(Student s) {

        if (s == null) {

            throw new IllegalArgumentException("Student cannot be null");

        }

        if (students == null) {

            students = new ArrayList<>(); // Initialize the list if it's null

        }

        if (students.contains(s)) {

            throw new IllegalStateException("Student already enrolled in this course: " + s.getStudentId());

        }

        this.students.add(s);

        System.out.println("Student " + s.getStudentId() + " is successfully enrolled in " + courseName);

    }

    public void removeStudent(Student s) {

        if (students.contains(s)) {

            this.students.remove(s);

            System.out.println("Student " + s.getStudentId() + " is successfully removed from " + courseName);

        } else {

            System.out.println("Student with ID " + s.getStudentId() + " not found in the list.");

        }

    }

    public void averageGrade() {

        if (students.isEmpty()) {

            System.out.println("No students enrolled in " + courseName);

            return;

        }

        Double sum = 0.0;

        for (Student student : students) {

            sum += student.getMarks();

        }

        Double average = sum / students.size();

        System.out.println("Average grade for" + courseName + " :" + average);

    }

    public Double calculateMedianGrade() {

        if (students.isEmpty()) {

            throw new IllegalStateException("No students enrolled in " + courseName + ". Cannot calculate median.");

        }

        ArrayList<Double> grades = new ArrayList<>();

        for (Student student : students) {

            grades.add(student.getMarks());

        }

        Collections.sort(grades);

        int size = grades.size();

        if (size % 2 == 1) {

            // Odd number

            return grades.get(size / 2);

        } else {

            // Even number

            return (grades.get(size / 2 - 1) + grades.get(size / 2)) / 2.0;

        }

    }

    @Override

    public String toString() {

        return "Course [courseName=" + courseName + ", courseId=" + courseId + ", credits=" + credits + "]";

    }

}

class Teacher extends person {

    // . Attributes: teacherID, name, specialization, list of courses taught.

    private String teacherId;

    private String teacherName;

    private String specialization;

    private ArrayList<Course> courses;

    public Teacher() {

    }

    public Teacher(String name, String email, LocalDate dateOfBirth, String ad, String teacherId,

            String teacherName, String specialization, ArrayList<Course> courses) {

        super(name, email, dateOfBirth, ad);

        this.teacherId = teacherId;

        this.teacherName = teacherName;

        this.specialization = specialization;

        this.courses = courses;

        University.TeacherList.add(this);

    }

    public String getTeacherId() {

        return teacherId;

    }

    public void setTeacherId(String teacherId) {

        this.teacherId = teacherId;

    }

    public String getTeacherName() {

        return teacherName;

    }

    public void setTeacherName(String teacherName) {

        this.teacherName = teacherName;

    }

    public String getSpecialization() {

        return specialization;

    }

    public void setSpecialization(String specialization) {

        this.specialization = specialization;

    }

    public ArrayList<Course> getCourses() {

        return courses;

    }

    public void setCourses(ArrayList<Course> courses) {

        this.courses = courses;

    }

    public void assignCourse(Course c) {

        this.courses.add(c);

        System.out.println("The Course " + c.getCourseName() + " has been assigned to : " + teacherName);

    }

    public void displayCourse() {

        System.out.println("The " + teacherName + " has been assigned to the following courses:");

        // Check if the teacher is assigned to any courses

        if (courses.isEmpty()) {

            System.out.println("No courses assigned.");

        } else {

            for (Course course : courses) {

                System.out.println("Course Name: " + course.getCourseName() +

                        " | Course ID: " + course.getCourseId());

            }

        }

    }

    public String toString() {

        return "Teacher [name=" + name + ", email=" + email + ", teacherId=" + teacherId + ", teacherName="

                + teacherName + ", specialization=" + specialization + "]";

    }

}

class AdministrativeStaff extends person implements Reportable {

    // Attributes: staffID, name, role, and department.

    private String staffID;

    private String role;

    private String department;

    public AdministrativeStaff(String name, String email, LocalDate dateOfBirth, String ad, String staffID, String role,

            String department) {

        super(name, email, dateOfBirth, ad);

        this.staffID = staffID;

        this.role = role;

        this.department = department;

        University.StaffList.add(this);

    }

    public String getStaffID() {

        return staffID;

    }

    public void setStaffID(String staffID) {

        this.staffID = staffID;

    }

    public String getRole() {

        return role;

    }

    public void setRole(String role) {

        this.role = role;

    }

    public String getDepartment() {

        return department;

    }

    public void setDepartment(String department) {

        this.department = department;

    }

    public void generateReport() {

        if (University.StudentList == null || University.CourseList == null) {

            throw new NullPointerException("People or courses list cannot be null");

        }

        System.out.println("Report Summary:");

        System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        System.out.println("Total Students: " + University.totalStudentsEnrolled());

        System.out.println("Total Teachers: " + University.totalTeachers());

        System.out.println("Total Administrative Staff: " + University.StaffList.size());

        System.out.println("Total Courses: " + University.totalCourses());

        System.out.println("\nCourse Details:");

        for (Course course : University.CourseList) {

            System.out.println("Course Name: " + course.getCourseName() +

                    ", Total Students Who Enrolled in this Course: " + course.getStudents().size() +

                    ", Teacher Who is Teaching  this Course: " + course.getTeacher().getName() +

                    ", Course ID: " + course.getCourseId() +

                    ", Credits: " + course.getCredits());

        }

    }

    public String toString() {

        return "AdministrativeStaff [name=" + name + ", email=" + email + ", staffID=" + staffID + ", role=" + role

                + ", department=" + department + "]";

    }

}

public class semproject1 {

    public static void main(String[] args) {

        try {

            // Example setup

            ArrayList<Student> students = new ArrayList<>();

            ArrayList<Teacher> teachers = new ArrayList<>();

            ArrayList<AdministrativeStaff> staff = new ArrayList<>();

            ArrayList<Course> courses = new ArrayList<>();

            University university = new University("Tech Campus", students, teachers, staff, courses);

            new crud();

            ArrayList<?> dataList = University.loadData("university\_data.txt"); // Load data as a generic list

        // Separate the data into specific lists

        ArrayList<Student> student1 = new ArrayList<>();

        ArrayList<Teacher> teacher1 = new ArrayList<>();

        ArrayList<AdministrativeStaff> staffs = new ArrayList<>();

        ArrayList<Course> course1 = new ArrayList<>();

        // Iterate through the loaded dataList and separate them by type

        for (Object item : dataList) {

            if (item instanceof Student) {

                student1.add((Student) item);

            } else if (item instanceof Teacher) {

                teacher1.add((Teacher) item);

            } else if (item instanceof AdministrativeStaff) {

                staffs.add((AdministrativeStaff) item);

            } else if (item instanceof Course) {

                course1.add((Course) item);

            }

        }

        // Now, you can use these lists to process or display the data as needed

        System.out.println("Students:");

        for (Student student : student1) {

            System.out.println(student);

        }

        System.out.println("Teachers:");

        for (Teacher teacher : teacher1) {

            System.out.println(teacher);

        }

        System.out.println("Administrative Staff:");

        for (AdministrativeStaff admin : staffs) {

            System.out.println(admin);

        }

        System.out.println("Courses:");

        for (Course course : course1) {

            System.out.println(course);

        }

        ArrayList<Course> course2 = University.getCourseList();

        System.out.println(course2);

        university.generateReport();

        } catch (IllegalArgumentException e) {

            System.err.println("Invalid argument: " + e.getMessage());

        } catch (IllegalStateException e) {

            System.err.println("Illegal operation: " + e.getMessage());

        } catch (Exception e) {

            // Catch-all for unexpected errors

            System.err.println("An unexpected error occurred: " + e.getMessage());

            e.printStackTrace();

        } finally {

            System.out.println("Program execution completed.");

        }

    }

}

University.java File

import java.util.ArrayList;

import java.io.\*;

public class University implements Reportable {

    private String campusName;

    public static ArrayList<Student> StudentList;

    public static ArrayList<Teacher> TeacherList;

    public static ArrayList<AdministrativeStaff> StaffList;

    public static ArrayList<Course> CourseList;

    public University(String campusName, ArrayList<Student> studentList, ArrayList<Teacher> teacherList,

            ArrayList<AdministrativeStaff> staffList, ArrayList<Course> courseList) {

        this.campusName = campusName;

        StudentList = studentList;

        TeacherList = teacherList;

        StaffList = staffList;

        CourseList = courseList;

    }

    public String getCampusName() {

        return campusName;

    }

    public void setCampusName(String campusName) {

        this.campusName = campusName;

    }

    public static ArrayList<Student> getStudentList() {

        return StudentList;

    }

    public static void setStudentList(ArrayList<Student> studentList) {

        StudentList = studentList;

    }

    public static ArrayList<Teacher> getTeacherList() {

        return TeacherList;

    }

    public static void setTeacherList(ArrayList<Teacher> teacherList) {

        TeacherList = teacherList;

    }

    public ArrayList<AdministrativeStaff> getStaffList() {

        return StaffList;

    }

    public void setStaffList(ArrayList<AdministrativeStaff> staffList) {

        StaffList = staffList;

    }

    public static ArrayList<Course> getCourseList() {

        return CourseList;

    }

    public static void setCourseList(ArrayList<Course> courseList) {

        CourseList = courseList;

    }

    public static void saveData(String filename) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) {

            // Save the last student, if present

            if (!StudentList.isEmpty()) {

                Student student = StudentList.get(StudentList.size() - 1);

                writer.write("Student|Name:" + student.getName() + "|Email:" + student.getEmail() + "|Address:" + student.getAddress() + "|Student ID:" + student.getStudentId() + "|Marks:" + student.getMarks() + "\n");

            }

            // Save the last teacher, if present

            if (!TeacherList.isEmpty()) {

                Teacher teacher = TeacherList.get(TeacherList.size() - 1);

                writer.write("Teacher|Name:" + teacher.getName() + "|Email:" + teacher.getEmail() + "|Address:" + teacher.getAddress() + "|Teacher ID:" + teacher.getTeacherId() + "|Specialization:" + teacher.getSpecialization() + "\n");

            }

            // Save the last administrative staff, if present

            if (!StaffList.isEmpty()) {

                AdministrativeStaff staff = StaffList.get(StaffList.size() - 1);

                writer.write("AdministrativeStaff|Name:" + staff.getName() + "|Email:" + staff.getEmail() + "|Address:" + staff.getAddress() + "|Staff ID:" + staff.getStaffID() + "|Role:" + staff.getRole() + "|Department:" + staff.getDepartment() + "\n");

            }

            // Save the last course, if present

            if (!CourseList.isEmpty()) {

                Course course = CourseList.get(CourseList.size() - 1);

                writer.write("Course|Name:" + course.getCourseName() + "|ID:" + course.getCourseId() + "|Credits:" + course.getCredits() + "\n");

            }

            System.out.println("Last elements successfully saved to " + filename);

        } catch (IOException e) {

            System.err.println("Error saving data: " + e.getMessage());

        }

    }

    public static <T> ArrayList<T> loadData(String filename) {

        ArrayList<T> dataList = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

            String line;

            while ((line = reader.readLine()) != null) {

                String[] parts = line.split("\\|");

                if (line.startsWith("Student")) {

                    String name = parts[1].split(":")[1];

                    String email = parts[2].split(":")[1];

                    String address = parts[3].split(":")[1];

                    String studentId = parts[4].split(":")[1];

                    double marks = Double.parseDouble(parts[5].split(":")[1]);

                    Student student = new Student(name, email, null, address, studentId, marks, null);

                    dataList.add((T) student);

                } else if (line.startsWith("Teacher")) {

                    String name = parts[1].split(":")[1];

                    String email = parts[2].split(":")[1];

                    String address = parts[3].split(":")[1];

                    String teacherId = parts[4].split(":")[1];

                    String specialization = parts[5].split(":")[1];

                    Teacher teacher = new Teacher(name, email, null, address, teacherId, null, specialization, null);

                    dataList.add((T) teacher);

                } else if (line.startsWith("AdministrativeStaff")) {

                    String name = parts[1].split(":")[1];

                    String email = parts[2].split(":")[1];

                    String address = parts[3].split(":")[1];

                    String staffID = parts[4].split(":")[1];

                    String role = parts[5].split(":")[1];

                    String department = parts[6].split(":")[1];

                    AdministrativeStaff staff = new AdministrativeStaff(name, email, null, address, staffID, role, department);

                    dataList.add((T) staff);

                } else if (line.startsWith("Course")) {

                    String courseName = parts[1].split(":")[1];

                    String courseId = parts[2].split(":")[1];

                    double credits = Double.parseDouble(parts[3].split(":")[1]);

                    Course course = new Course(courseName, courseId, credits, new ArrayList<>(), null);

                    dataList.add((T) course);

                }

            }

            System.out.println("Data successfully loaded from " + filename);

        } catch (IOException e) {

            System.err.println("Error loading data: " + e.getMessage());

        }

        return dataList;

    }

    // Helper method to find a Course by its name

    private static Course findCourse(String courseName) {

        for (Course course : CourseList) {

            if (course.getCourseName().equals(courseName)) {

                return course;

            }

        }

        return null; // Return null if not found

    }

    // Helper method to find a Teacher by name

    private static Teacher findTeacher(String teacherName) {

        for (Teacher teacher : TeacherList) {

            if (teacher.getName().equals(teacherName)) {

                return teacher;

            }

        }

        return null; // Return null if not found

    }

    // Other methods remain unchanged...

    public static void addStudent(Student s) {

        if (s == null) {

            throw new IllegalArgumentException("Student cannot be null");

        }

        if (StudentList.contains(s)) {

            throw new IllegalStateException("Student already enrolled in this course: " + s.getStudentId());

        }

        StudentList.add(s);

        System.out.println("Student " + s.getStudentId() + " is successfully enrolled in University ");

    }

    public static void removeStudent(Student s) {

        if (s == null) {

            throw new IllegalArgumentException("Student cannot be null");

        }

        if (!StudentList.contains(s)) {

            throw new IllegalStateException("Student does not Exist");

        }

        StudentList.remove(s);

        System.out.println("Student " + s.getStudentId() + " is successfully removed in University ");

    }

    public static int totalStudentsEnrolled() {

        return StudentList.size();

    }

    public static int totalTeachers() {

        return TeacherList.size();

    }

    public static int totalCourses() {

        return CourseList.size();

    }

    public void generateReport() {

        if (University.StudentList == null || University.CourseList == null) {

            throw new NullPointerException("People or courses list cannot be null");

        }

        System.out.println("Report Summary:");

        System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        System.out.println("Total Students: " + University.totalStudentsEnrolled());

        System.out.println("Total Teachers: " + University.totalTeachers());

        System.out.println("Total Administrative Staff: " + University.StaffList.size());

        System.out.println("Total Courses: " + University.totalCourses());

    }

}